Prepartition: A New Paradigm for the Load Balance of Virtual Machine Reservations in Data Centers

Wenhong Tian*, Minxian Xu, Yu Chen, Yong Zhao
School of Computer Science and Engineering
University of Electronic Science and Technology of China, Chengdu, China
Email: tian_wenhong@uestc.edu.cn

***Abstract*—It is significant to apply load-balancing strategy to improve the performance and reliability of resource in data centers. One of the challenging scheduling problems in Cloud data centers is to take the allocation and migration of reconfigurable virtual machines (VMs) as well as the integrated features of hosting physical machines (PMs) into consideration. In the reservation model, workload of data centers has fixed process interval characteristics. In general, load-balance scheduling is NP-hard problem as proved in many open literatures. Traditionally, for offline load balance without migration, one of the best approaches is LPT (Longest Process Time first), which is well known to have approximation ratio 4/3. With virtualization, reactive (post) migration of VMs after allocation is one popular way for load balance and traffic consolidation. However, reactive migration has difficulty to reach predefined load balance objectives, and may cause interruption and instability of service and other associated costs. In view of this, we propose a new paradigm-Prepartition: it proactively sets process-time bound for each request on each PM and prepares in advance to migrate VMs to achieve the predefined balance goal. Prepartition can reduce process time by preparing VM migration in advance and therefore reduce instability and achieve better load balance as desired. Trace-driven and synthetic simulation results show that Prepartition has 10%-20% better performance than the well known load balancing algorithms with regard to average CPU utilization, makespan as well as capacity_makespan.**

***Index Terms*—Cloud Computing; Physical Machines (PMs); Virtual Machines (VMs); Reservation Model; Load Balance Scheduling**

I. Introduction

In traditional data centers, applications are tied to specific physical servers that are often over-provisioned to deal with upper-bound workload. Such configuration makes data centers expensive to maintain with wasted energy and floor space, low resource utilization and significant management overhead. With virtualization technology, today's Cloud data centers become more flexible, secure and provide better support for on-demand allocating. The definition and model defined by this paper are aimed to be general enough to be used by a variety of Cloud providers and focus on the Infrastructure as a Service (IaaS). Cloud datacenters can be a distributed network in structure, containing many compute nodes (such as servers), storage nodes, and network devices. Each node is formed by a series of resources such as CPU, memory, and network bandwidth and so on, which are called multi-dimensional resources; each has its corresponding properties. Under virtualization, Cloud data centers should have ability to migrate an application from one set of resources to another in a non-disruptive manner. Such ability is essential in modern cloud computing infrastructure that aims to efficiently share and manage extremely large data centers. Reactive migration of VMs is widely proposed for load balance and traffic consolidation.

One key technology playing an important role in Cloud data centers is load balance scheduling. There are quite many load balance scheduling algorithms. Most of them is for traditional web servers but does not consider VM reservations with lifecycle characteristics. One of the challenging scheduling problems in Cloud data centers is to consider allocation and migration of reconfigurable VMs and integrated features of hosting PMs. The load balance problem for VM reservations considering lifecycle is as follows: given a set of m identical machines (PMs) $PM_1, PM_2, \ldots, PM_m$ and a set of n requests (VMs), each request [s_i, f_i, d_i], has a start-time (s_i), end-time (f_i) constraint and a capacity demand (d_i) from a PM, the objective of load balance is to assign each request to one of PMs so that the loads placed on all machines are balanced or the maximum load is minimized. This problem is not studied yet in the open literatures. The major contributions of this paper are:

- Providing a modeling approach to VM reservation scheduling with capacity sharing by modifying tra-

978-1-4799-2003-7/14/$31.00 ©2014 IEEE

ditional interval scheduling problem and considering life-cycles characteristics of both VMs and PMs.
- Designing and implementing a load balancing scheduling algorithm–Prepartition which can prepare migration in advance and set process time bound for each VM on a PM. Computational complexity and approximation analysis are also provided for Prepartition.
- Providing performance evaluation of multiple metrics such as makespan, average utilization, total migration numbers as well as capacity_makespan. by simulating different algorithms using trace-driven and synthetic data.

The remaining parts of this paper are organized as follows. Section II discusses the related work on load balance algorithms. Section III introduces problem formulation. Section IV presents Prepartition algorithm in details. Performance evaluation of different scheduling algorithms is shown in section V. Finally in section VI, a conclusion is given.

II. Related works

A great mount of work has been devoted to the schedule algorithms and can be mainly divided into two types: online load balance algorithms and offline ones. The major difference lies in that online schedulers only know current request and status of all PMs but offline schedulers know all the requests and status of all PMs. Armbrust et al.[1] summarized the key issues and solutions in Cloud computing. Foster et al.[3] provided detailed comparison between Cloud computing and Grid computing. Wood et al.[11] introduced techniques for virtual machine migration with spots and proposed a few reactive migration algorithms. Arzuaga et al.[2] proposed a quantifying measure of load imbalance on virtualized enterprise servers considering reactive live VM migrations. Gulati et al [5] presented challenge issues and Distributed Resource Scheduling (DRS) as a load balance scheduling for Cloud-scale resource management in VMware. Tian et al. [8] design a toolkit for modeling and simulating VM allocation, [9][10] introduced a dynamic load balance scheduling algorithm considering only current allocation period and multi-dimensional resource but without considering life-cycles of both VMs and PMs. Most of existing research does not consider fixed interval constraints of VM allocation. Knauth et al. [6] introduce energy-efficient scheduling algorithms applying timed instances that have an a priori specified reservation time of fixed length by following divisible capacity configuration, these assumptions are also adopted in this paper. Most of existing research considers reactive VM migrations as a mean for load balance in data centers. To the best of our knowledge, proactive VM migration by pre-partition has not been studied yet in the open literatures. This is one of major objectives in this paper.

III. Problem Formulation

A. Problem description and formulation

In this paper we consider VMs reservation and model the VM allocations as a modified interval scheduling problem (MISP) with fixed processing time. More explanation and analysis about traditional interval scheduling problems with fixed processing time can be found in [7] and references there in. We present a general formulation of modified interval-scheduling problem and evaluate its results compared to well-known existing algorithms. There are following assumptions:

1) All data are deterministic and unless otherwise specified, the time is formatted in slotted windows. we partition the total time period [0, T] into slots with equal length (s_0), the total number of slots is k=T/s_0. The start time s_i and finish time f_i are integer numbers of one slot. Then the interval of a request can be represented in slot format with (start-time, finish-time). For example, if s_0=5 minutes, an interval (3, 10) means that it has start time and finish time at the 3rd-slot and 10th-slot respectively. The actual duration of this request is (10-3)$\times$5=35 minutes.

2) For all VM reservations, there are no precedence constraints other than those implied by the start-time and finish-time.

3) The required capacity of each request is a positive real number between (0,1]. Notice that the capacity of a single physical machine is normalized to be 1 and the required capacity of a VM can be 1/8, 1/4 or 1/2 or other portions of the total capacity of a PM. This is consistent with widely adopted practice in Amazon EC2 [13] and [6].

A few key definitions are explained as follows:

Traditional interval scheduling problem (ISP) with fixed processing time: A set of requests $\{1, 2,\ldots, n\}$ where the i-th request corresponds to an interval of time starting at s_i and finishing at f_i , each request needs a capacity of 1, i.e. occupying the whole capacity of a machine during fixed processing time.

Interval scheduling with capacity sharing (ISWCS): The only difference from traditional interval scheduling is that a resource (to be concrete, a PM) can be shared by different requests if the total capacity of all requests allocated on the single resource at any time does not surpass the total capacity that the resource can provide.

Sharing compatible intervals for ISWCS: A subset of intervals with total required capacity not surpass the total capacity of a PM at any time, therefore they can share the capacity of a PM. In the literature, the makespan is used to measure the load balance, which is simply the maximum total load (processing time) on any machine. Traditionally, the makespan is the total length of the schedule (that is, when all the jobs have finished processing where each job occupies the whole capacity of a machine during processing).

In view of the problem in ISWCS, we redefine the makespan as capacity_makespan.

Capacity_makespan of a PM i: In any allocation of VM requests to PMs, let $A(i)$ denote the set of VM requests allocated to machine PM_i. Under this allocation, machine PM_i will have total load equal to the sum of product of each required capacity and its duration (called Capacity_makespan, i.e., CM for abbreviation in this paper), as follows:

$$CM_i = \sum_{j \in A(i)} d_j t_j \tag{1}$$

where d_j is the capacity requests of VM_j from a PM and t_j is the span of request j (i.e., the length of processing time of request j).

Therefore, the goal of load balancing is to minimize the maximum load (capacity_makespan) on any PM. Some other related metrics such as average utilization and makespan are also considered and will be explained in the following section. Assuming there are m PMs in data centers, the problem of ISWCS load balance in it therefore can be formulated as:

$$Min_{i=1}^{m} CM_i \tag{2}$$

$$\textbf{subject to 1).} \ \forall \ slot \ s, \sum_{VM_j \in PM_i} d_j \leq 1 \tag{3}$$

$$\textbf{2).} \ \forall \ j, s_j \ and \ e_j \ are \ fixed \ by \ reservation. \tag{4}$$

where d_j is the capacity requirement of VM j and the total capacity of a PM i is normalized to 1. The condition 1) shows the sharing capacity constraint and condition 2) is for the interval constraint of VM reservations.

Theorem 1 The offline scheduling problem of finding an allocation of minimizing the makespan in general case is NP-complete.

The proof can be found in [10] and is omitted here.

B. Metrics for ISWCS load balancing algorithms

In this section, a few metrics closely related to ISWCS load balance problem will be presented. Some other metrics can be found in Tian et al. [9][10].

1) PM resource: $PM_i(i, PCPU_i, PMem_i, PSto_i)$, i is the index number of PM, $PCPU_i$,$PMem_i$,$PSto_i$ are the CPU, memory, storage capacity of that a PM can provide.

2) VM resource:
$VM_j(j, VCPU_j, VMem_j, VSto_j, T_j^{start}, T_j^{end})$, j is the VM type ID, $VCPU_j, VMem_j, VSto_j$ are the CPU, memory, storage requirements of VM_j, T_j^{start}, T_j^{end} are the start time and end time, which are used to represent the life cycle of a VM.

3) Time slots: we consider a time span from 0 to T be divided into slots with same length. The n slots can be defined as $[(t_1 - t_0), (t_2 - t_1), \ldots, (t_n - t_{n-1})]$, each time slot T_k means the time span $(t_k - t_{k-1})$.

4) Average CPU utilization of PM_i during slot 0 and T_n is defined as:

$$PCPU_i^U = \frac{\sum_{k=0}^{n} (PCPU_i^{T_k} \times T_k)}{\sum_{k=0}^{n} T_k} \tag{5}$$

where $PCPU_i^{T_k}$ is the average CPU utilization during slot T_k. Average memory utilization ($PMem_i^U$) and storage utilization ($PSto_i^U$) of both PMs can be computed in the same way. Similarly, average CPU (memory and storage) utilization of a VM can be computed.

5) Makespan: is the total length of a schedule for a set of VM reservations, i. e., when all the jobs have finished processing.

6) The capacity_makespan (CM) of all PMs: can be formulated as:

$$CM = \max_i (CM_i) \tag{6}$$

From these equations, we notice that life cycle and capacity sharing are two major differences from traditional metrics such as makespan which only considers process time (duration). Traditionally Longest Process Time first (LPT) [4] is widely used for load balance of offline multi-processor scheduling. Reactive (post) migration of VMs is another popular way of load-balancing. *However, reactive migration has difficulty to reach predefined load balance objectives, and may cause interruption and instability of service and other associated costs*. By considering both fixed process intervals and capacity sharing properties in Cloud data centers, we propose a new offline algorithm–Preparitition as follows.

IV. PREPARTITION ALGORITHM

For a given set of VM reservations, let us consider there are m PMs in a data center and denote OPT as the optimal solution for a given set of J VM reservations.

Firstly define

$$P_0 = max\{max_{j=1}^{J} CM_j, \frac{1}{m}\sum_{j=1}^{J} CM_j\} \leq OPT \quad (7)$$

P_0 is a lower bound on OPT. Fig.2 shows the pseudocodes of Prepartition algorithm. The algorithm firstly computes balance value by equation (7), defines partition value (k) and finds the length of each partition (i.e. $\lceil P_0/k \rceil$, which is the max time length a VM can continuously run on a PM). For each request, Preparition equally partitions it into multiple $\lceil P_0/k \rceil$ subintervals if its CM is larger than $\lceil P_0/k \rceil$, and then finds a PM with the lowest average capacity_makespan and available capacity, and updates the load on each PM. After all requests are allocated, the algorithm computes the capacity_makespan of each PM and finds total partition (migration) numbers. For practice, the scheduler has to record all possible subintervals and their hosting PMs of each request so that migrations of VMs can be conducted in advance to reduce overheads.

Input: VM requests indicated by their (required VM type IDs, start times, ending times, requested capacity), CM_i is the capacity_makespan of request *i*.

Output: Assign a PM ID to all requests and their partitions (migrations)
Initialization: computing the bound P_0 value, set the partition value *k*.

1: IF $CM_i > (P_0/k)$, *THEN partitions it into multiple* P_0/k *subintervals equally* and consider each subinterval as a new request
2: Sorts all intervals in decreasing order of *CMs*, break ties arbitrarily;
3: Let $I_1, I_2, ..., I_n$ denote the intervals in this order
3: **For** *j* from *1* to *n* ***Do***
4: allocates *j* to the PM with the lowest load and available capacity
5: updates load (*CM*) of the PM
6: **Endfor**
7: Computes *CM* of each PM and total partitions (migrations)

Fig. 1. The pseudo codes of Prepartition algorithm

Theorem 2: The computational complexity of Prepartition algorithm is $O(nlogm)$ *using priority queue data structure where* n *is the number of VM requests after pre-partition and* m *is total number of PMs used.*

Proof: The priority queue is designed such that each element (PM) has a priority value (average capacity_makespan), and each time the algorithm needs to select an element from it, the algorithm takes the one with highest priority (the smaller average capacity_makespan value is, the higher priority it is). Sorting a set of n number in a priority queue takes $O(n)$ time and a priority queue performs insertion and the extraction of minima in $O(logn)$ steps (detailed proof of the priority queue is shown in [7]). Therefore, by using priority queue or related data structure, the algorithm can find a PM with the lowest average capacity_makespan in $O(logm)$ time. Altogether, for n requests, Prepartition algorithm has time complexity $O(nlogm)$.

Theorem 3: The approximation ratio of Prepartition algorithm is $(1+\epsilon)$ *where* ϵ=$\frac{1}{k}$.

Proof: This is because that each request has bounded capacity_makespan by pre-parition based on ideal lower bound P_0. We sketch the proof as follows. Each job has start-time s_i, end-time (f_i) and process time p_i=f_i-s_i. Considering the last job to finish (after scheduling all other jobs) and supposing this job starts at time T_0. All the machines must have been fully loaded up to T_0, which gives $T_0 \leq$OPT. Since, for all jobs, we have $p_i \leq \epsilon$ OPT (by setting of Prepartition algorithm), this job finishes at T_0+ϵOPT. Hence, the schedule can be no more than T_0+ϵ OPT $\leq$ (1+ϵ)OPT, this finishes the proof.

V. Performance Evaluation

In this part, we will present the simulation results between Prepartition algorithm and other three existed algorithms. To achieve this goal, we used a Java simulator CloudSched (see Tian et al. [8]). For simulation, to be realistic and reasonable, we adopt data both from normal distribution and Lawrence Livermore National Lab (LLNL) trace, see [12] for detailed introduction about the trace. All simulations are conducted on a computer configured with Intel i5 processor at 2.5GHz and 4GB memory. Round-Robin (RR) algorithm, Longest Process Time (LPT) algorithm and Post Migration Algorithm (PMG) are also implemented:

1) Round-Robin Algorithm: a traditional load balancing scheduling algorithm by allocating the VM requests in turn to each PM that can provide required resource.

2) Longest Processing Time first (LPT): it sorts the VM requests by processing time in decreasing order firstly. Then allocating the requests in that order to the PM with the lowest load. In this paper, the lowest load means the lowest capacity_makespan of all PMs.

3) Post Migration algorithm (PMG): Firstly, it processes the requests in the same way as LPT does. Then the average capacity_makespan of all jobs is calculated. The up-threshold and low-threshold of the capacity_makespan for the post migration are calculated through the average capacity_makespan multiplied by a factor (in this

TABLE I
8 TYPES OF VIRTUAL MACHINES (VMS) IN AMAZON EC2

Compute Units	Memory	Storage	VM Type
1 units	1.7GB	160GB	1-1(1)
4 units	7.5GB	850GB	1-2(2)
8 units	15GB	1690GB	1-3(3)
6.5 units	17.1GB	420GB	2-1(4)
13 units	34.2GB	850GB	2-2(5)
26 units	68.4GB	1690GB	2-3(6)
5 units	1.7GB	350GB	3-1(7)
20 units	7GB	1690GB	3-2(8)

TABLE II
3 TYPES OF PHYSICAL MACHINES (PMS) SUGGESTED

PM Pool Type	Compute Units	Memory	Storage
Type 1	16 units	30GB	3380GB
Type 2	52 units	136GB	3380GB
Type 3	40 units	14GB	3380GB

paper we set the factor as 0.1, so the up-threshold is average capacity_makespan multiplied by 1.1 and the low-threshold is multiplied by 0.9). Of course the factor can be set dynamically to meet different requirements; however, the larger the factor is, the higher imbalance is. A migration list is formed by collecting the VMs taken from PMs with capacity_makespan higher than the low-threshold. The VMs would be taken from a PM only if the operation would not lead the capacity_makespan of the PM to be less than the low threshold. After that, the VMs in the migration list would be re-allocated to a PM with capacity_makespan less than the up-threshold. The VMs would be allocated to a new PM only if the operation would not lead the capacity_makespan of the PM to be higher than the up-threshold. There may be still some VMs left in the list, finally the algorithm allocates the left VMs to the PMs with the lowest capacity_makespan until the list is empty.

In this paper, we adopt the Amazon EC2 configuration of VMs and PMs as shown in Table I and II. Note that one compute unit (CU) has equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor [13].

Observation 1 PMG is a best-effort trial heuristic for load balance. It does not guarantee a bounded or predefined load balance objective. This is validated in the performance evaluation section.

A. replay with LLNL data traces

As for realistic data, we adopt the log data at Lawrence Livermore National Lab (LLNL) [12]. The log contains months of records collected by a large Linux cluster and has characteristics consistent with our problem model. Each line of data in that log file includes 18 elements, while we only need the request-ID, start-time, duration and number of processors (capacity demands) in our simulation. We convert the units from seconds in LLNL log file into minutes, because we set 5 minutes as a time slot length mentioned in previous section. Fig.2 and Fig.3

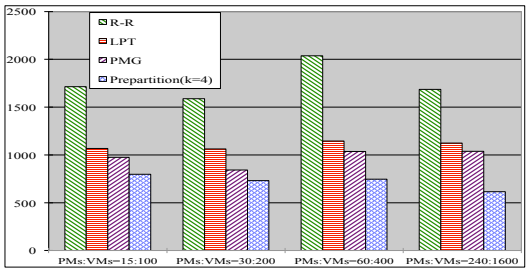

Fig. 2. The comparison of makespan with LLNL trace

show the makespan and capacity_makespan comparison for different algorithms with LLNL data traces. We observe that Prepartition algorithm has better performance than other algorithms in average utilization, makespan, capacity_makespan. Prepartition algorithm has 10%-20% higher average utilization than PMG and LPT, and 40%-50% higher average utilization than Random-Robin (RR). Prepartition algorithm has 10%-20% lower average makespan and capacity_makespan than PMG and LPT, and 40%-50% lower average makespan and capacity_makespan than Random-Robin (RR). Because of page limit, some similar results are omitted.

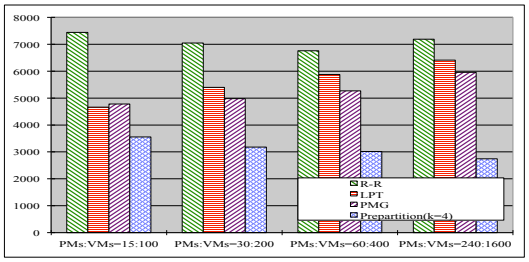

Fig. 3. The comparison of capacity_makespan of LLNL trace

Observation 2 Whatever numbers of migrations to taken, post migration algorithm (PMG) just cannot achieve the same level of average utilization, makespan and capacity_makespan as Prepartition does.

This is because that Prepartition works in a much more refined and desired scale by partition based on reservation data while PMG is just a best-effort trial by migration.

B. Results comparison by synthetic data

We set 5 minutes as a slot, so 12 slots are for an hour, 96 slots are for a day, 2880 slots are for a month. All requests satisfy the Normal distribution, with parameters mean μ and standard deviation δ as 288 (three days) and 96 (one day) respectively. For collecting data, we firstly fix the k value of Prepartition algorithm as 4. Different types of VMs have equal probability. Then we change the ratio of VMs to PMs numbers as $15:100$, $30:200$, $60:400$ and $240:1600$ to track the tendency. Each set of data is the average values of 10-runs. Fig. 4 and Fig. 5 show the makespan and

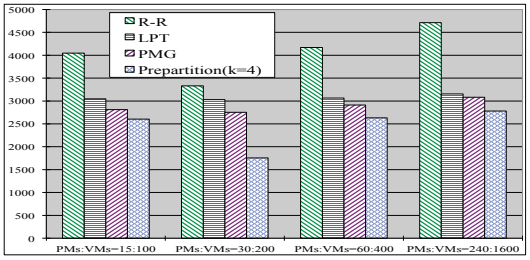

Fig. 4. The comparison of makespan-Normal distribution

capacity_makespan comparison of different algorithms respectively. We observe that Prepartition algorithm has 10%-20% higher average utilization than PMG and LPT, and 40%-50% higher average utilization than Random-Robin (RR); Prepartition algorithm has 8%-13% lower average makespan and capacity_makespan than PMG and LPT, and 40%-50% lower average makespan and capacity_makespan than Random-Robin (RR). Because of page limit, some similar results are omitted.

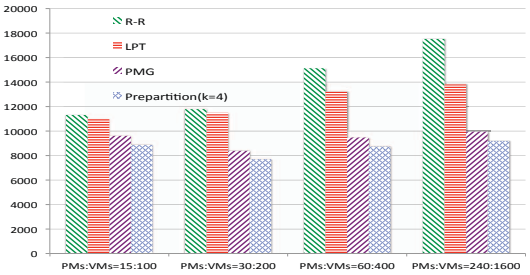

Fig. 5. The comparison of capacity_makespan of Normal distribution

VI. CONCLUSION

In this paper, to reflect the feature of capacity sharing and fixed interval constraint in Cloud data centers, we propose a new offline load-balancing algorithm Prepartition. Theoretically we prove that Prepartition is a $(1+\epsilon)$-approximation where $\epsilon=\frac{1}{k}$ and k is a positive integer. By increasing k it is possible to be very close to optimal solution, i.e., by setting k value, it is also possible to achieve predefined load balance goal as desired. There are still a few research issues such as making suitable choices between total partition numbers and load balance objective, analyzing the performance in a real data center, and considering precedence constraints among different VM requests.

Acknowledgement This research is partially supported by Central University Fund (ID-ZYGX2013J073), 2013 CCF-Tencent Open Research Fund (CCF-TencentAGR20130110) and China National Science Foundation (CNSF) with project ID 61150110486.

REFERENCES

[1] M. Armbrust et al., Above the Coulds: A Berkeley View of Cloud Computing, technical report,2009.
[2] E. Arzuaga, D. R. Kaeli, Quantifying load imbalance on virtualized enterprise servers, in the proceedings of WOSP/SIPEW 10, January 28-30, 2010, San Jose, California, USA.
[3] I. Foster, Y. ZHAO, I. RAICU, S. Lu, Cloud Computing and Grid Computing 360-Degree Compared, IEEE International Workshop on Grid Computing Environments (GCE) 2008, co-located with IEEE/ACM Supercomputing 2008.
[4] R. L. Graham 1969. *Bounds on Multiprocessing Timing Anomalies*, SIAM Journal on Applied Mathematics, Vol.17, No.2. (Mar., 1969), pp.416-429.
[5] A. Gulati, G. Shanmuganathan, A. Holler, I. Ahmad, Cloud-scale resource management: challenges and techniques, VMware Technical Journal, 2011.
[6] T. Knauth, C. Fetzer, Energy-aware Scheduling for Infrastructure Clouds, In the proceedings of CloudCom 2012.
[7] J. Kleinberg, E. Tardos, Algorithm Design, Pearson Education Inc., 2005.
[8] W. Tian, Y. Zhao, Y. Zhong, C. Jing, X. Sun, A Toolkit For Modeling and Simulation of Real-time Virtual Machine Allocation in a Cloud Data Center, IEEE Trans Automation Science and Engineering, DOI (identifier) 10.1109/TASE.2013.2266338, Online First, July of 2013.
[9] W. Tian, Y. Zhao,Y. Zhong, M. Xu, C. Jing, dynamic and integrated load-balancing scheduling algorithms for Cloud data-centers, China Communications, 2011, Vol. 8 Issue (6): 117-126.
[10] W. Tian, X. Liu, C. Jin, Y. Zhong, LIF: A dynamic scheduling algorithm for Cloud data centers considering multi-dimensional resources, Journal of Information & Computational Science, Aug. 12, 2013, 10:12,pp.3925-3937.
[11] T. Wood, et. al., Black-box and Gray-box Strategies for Virtual Machine Migration, in the proceedings of Symp. on Networked Systems Design and Implementation (NSDI), 2007.
[12] Hebrew University, Experimental Systems Lab, www.cs.huji.ac.il/labs/parallel/workload, 2013
[13] Amazon, Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2/, 2013